CMPT 365 Programming Assignment 2
Java Skeleton Program Tutorial

Mark Drew [mark@cs.sfu.ca]
Chen Song [csa102@sfu.ca]

1. Prerequisites

This tutorial assumes that you already know:

- **Fundamentals of Java programming**. You should know how to use variables and loops. You should be able to tell the difference between a class and an object. You should be able to know when and how to use public and private methods. You should also know what a package is.

You will have three different options if you do not know how to program in Java:

- Choose the C++ alternative if you understand the fundamentals of C++ programming.
- Learn Java. As long as you are familiar with any modern object orientated programming language, you should be able to pick up Java in several hours. I recommend the book "Introduction to Java Programming" written by Y. Daniel Liang. You will only need the first couple of chapters. Any edition is fine.
- Do this assignment in another language you are familiar with. You should discuss with the TA or the instructor about that.

This tutorial also assumes that you are using Windows. If you are a Linux user, you are definitely smart enough to figure out how to configure the environment yourself. On the other hand, I really hope that I can help Mac OS users, but unfortunately, I cannot afford a Mac.

Experience with GUI design tools is preferred (QT, Swing, GUIDE, graphics.h, …).

2. Environment Configurations

2.1 Install JDK

JDK stands for Java Development Kit, which includes the Java compiler and the Java Virtual Machine.

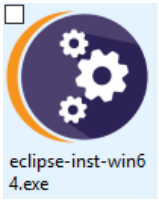
Search “jdk” on Google. You should be able to find the download link easily. The final executable file looks like this:

Depending on the date you read this tutorial, the version number “8u131” may be different. Double click the executable to install it.

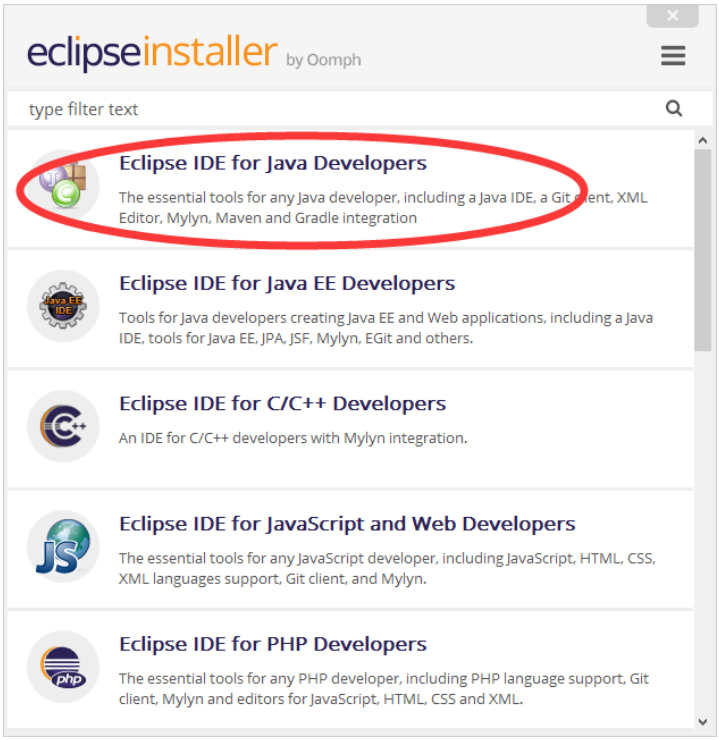
2.2 Install Eclipse

Eclipse is the Integrated Development Environment (IDE) we are going to use when writing code in Java.

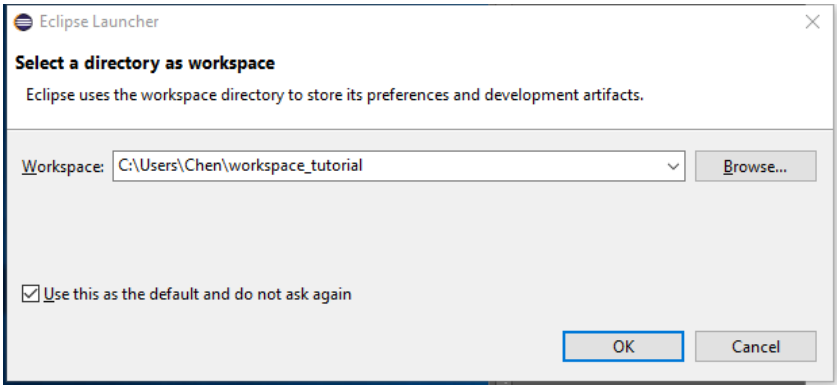
Search “eclipse” on Google. You should be able to find the download link easily. The final executable file looks like this:



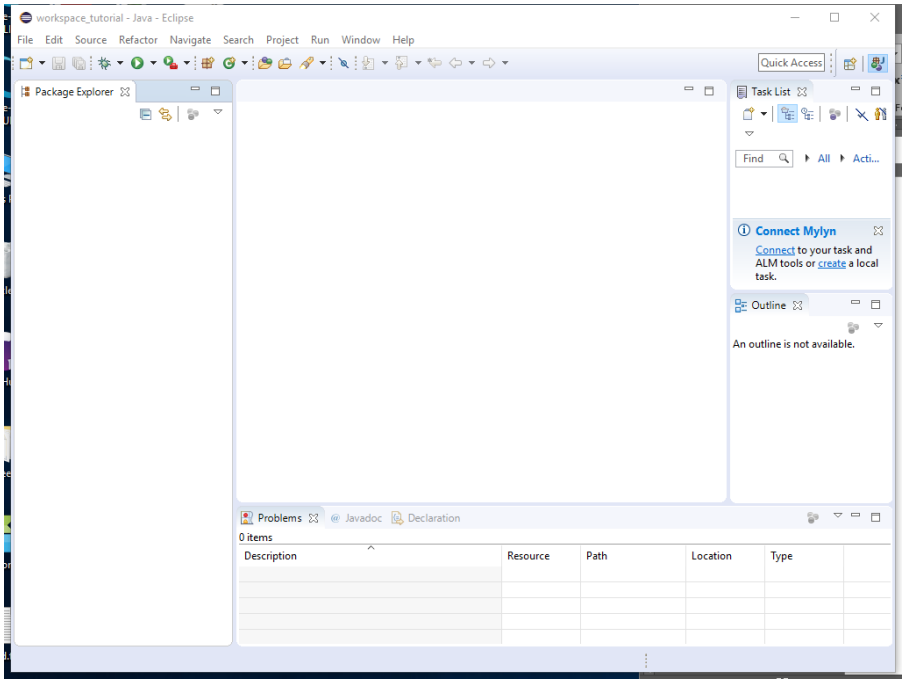
Double click the executable. Select “Eclipse IDE for Java Developers” and install eclipse.



Launch eclipse after your installation. You need to configure the workspace directory the first time you run eclipse. If you don’t know what it is, keep the default and select “OK”.



Click the “restore” button at the top-left of your screen, then you should expect a window like this:



Congratulations! Eclipse is now successfully set up.

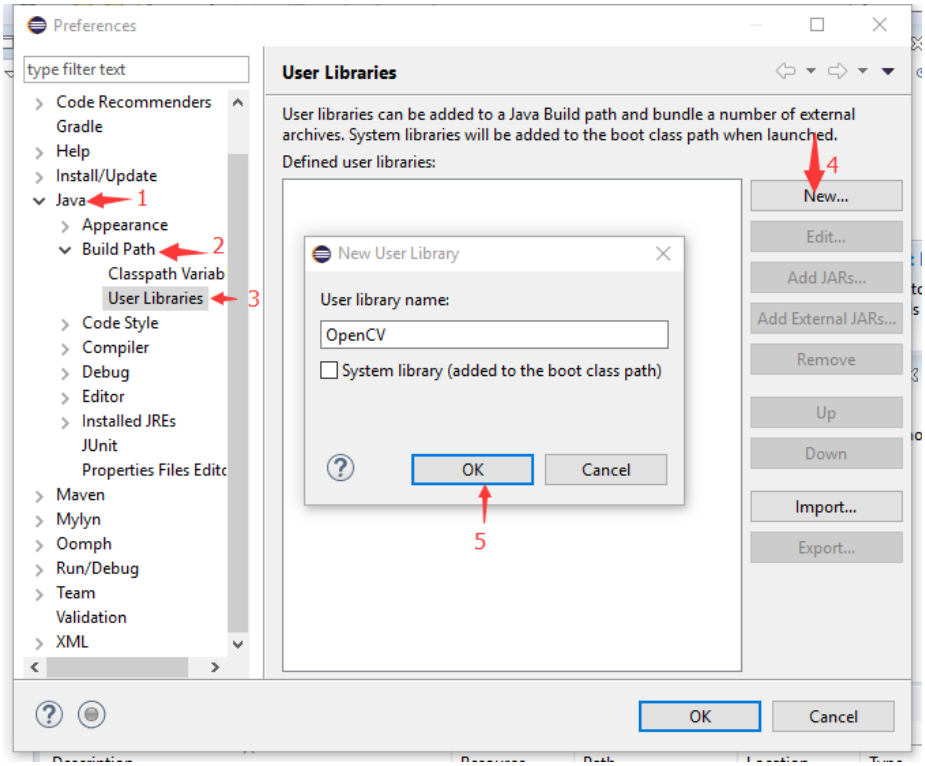
2.3 Install OpenCV

Visit http://opencv.org/releases.html and download “Win pack” under version 3.2.0. It should look like this:

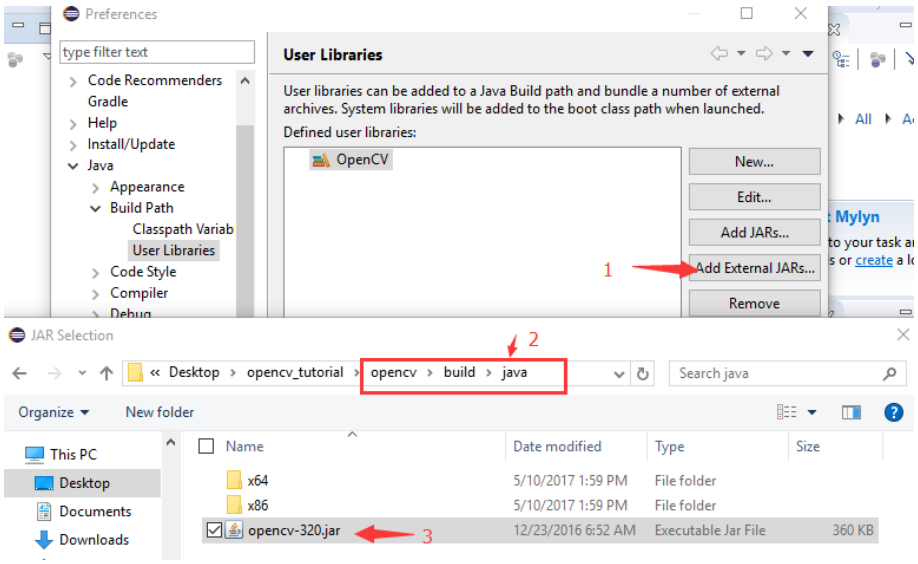
Double click the executable and extract the OpenCV package to your favorite directory. Note that this package includes not only OpenCV, but also FFMPEG. Therefore, it not only supports image but also video processing. Read the assignment specification again carefully if you have no idea about what OpenCV and FFMPEG are.

Please copy all files from “opencv\build\bin” to “opencv\build\java\x64” (or x86 if you are using a x86 operating system).

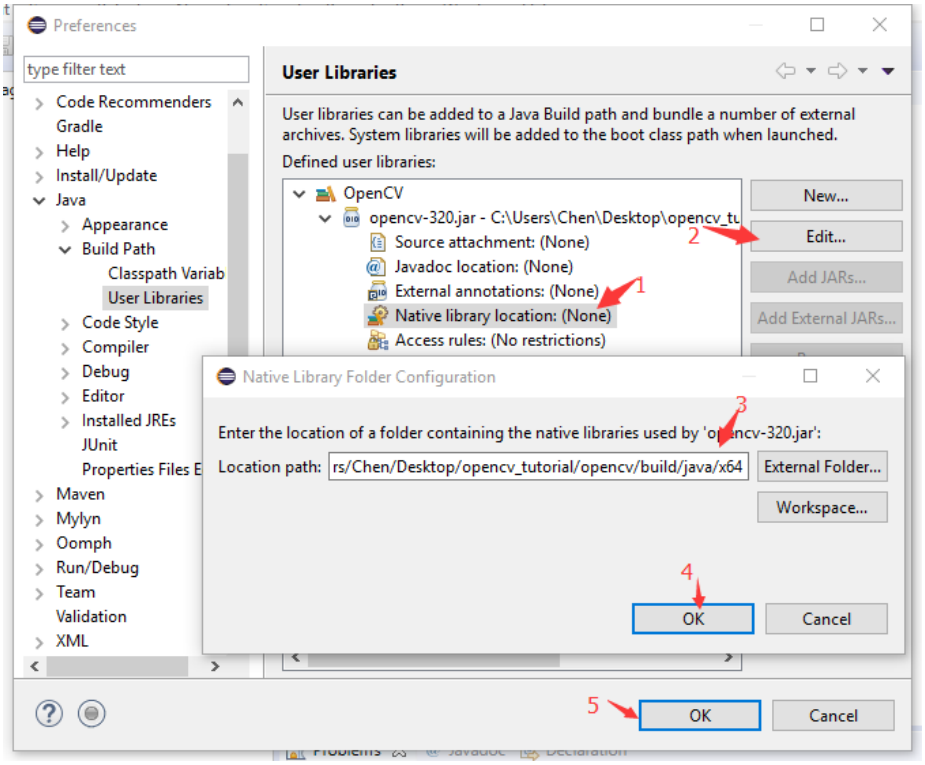
Now you want to inform eclipse of the location of OpenCV library. In eclipse's main window, click "Window -> Preferences". Then, in the pop-up window, select "Java -> Build Path -> User Libraries -> New". Enter your favorite name for the library and click "OK".



Select your new library ("OpenCV" in my case) and click "Add External JARs…". Then, select "<where you extract the OpenCV library>\opencv\build\java\opencv-320.jar".



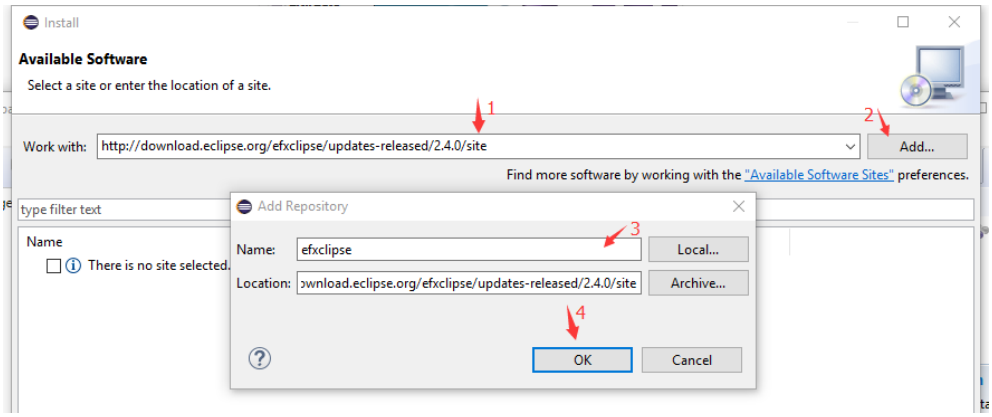
Select “Native library location: (None)” and click “Edit”. Enter “<where you extract the OpenCV library>\opencv\build\java\x64” (or x86 if you are using a x86 system) and click “OK”.



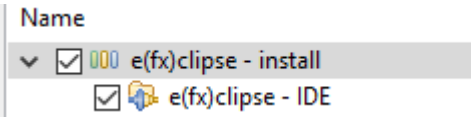
2.4 Install JavaFX

We are going to use JavaFX to create the Graphical User Interface (GUI).

In eclipse’s main window, click “Help -> Install New Software…”. In the pop-up window, enter “http://download.eclipse.org/efxclipse/updates-released/2.4.0/site” in the “work with:” text field and click “Add…”. In the pop-up window, enter your favorite name and then click “OK”.



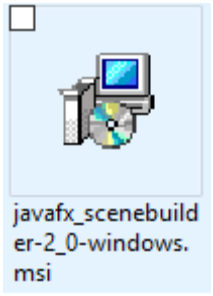
Wait until eclipse fetches all the information from the website. Select “e(fx)clipse - IDE” and click “Next”.



Accept the license to finish your installation.

2.5 Install JavaFX Scene Builder

JavaFX describes GUI in XML (you should already know what XML is), which is not very convenient in terms of editing. JavaFX scene builder frees us from having to type in XML code manually. Search “JavaFX Scene Builder” on Google and you should be smart enough to download the 2.0 version. The final file looks like this:



Double click the file to install scene builder.

3. Get Started

3.1 Link Libraries

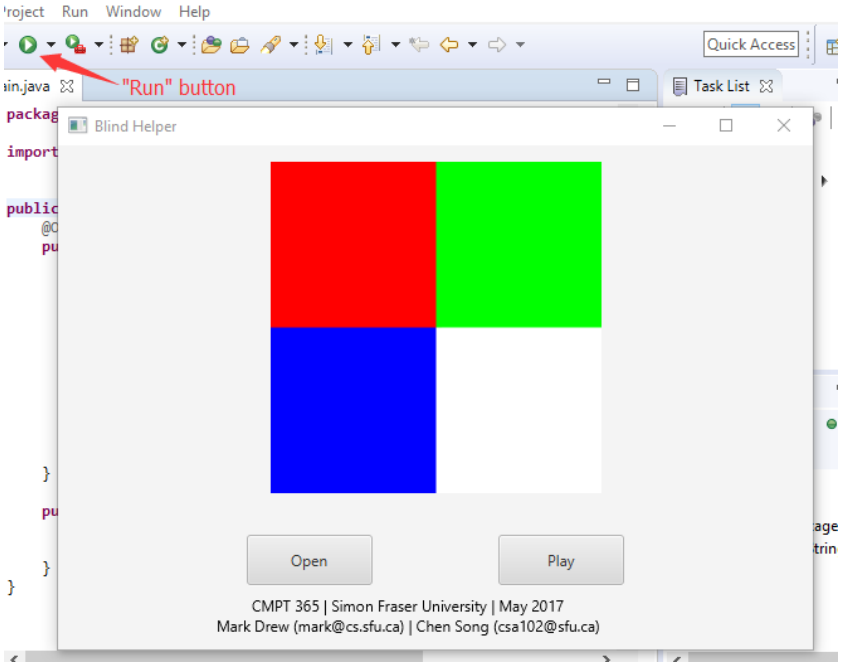
In eclipse's main window, import the existing skeleton code using "File -> Open Projects from File System…".

You are likely to find some error notifications once you open the project. This is because the libraries are not linked properly. Click "Project -> Properties". In the pop-up window, select "Java Build Path" then "Libraries". Remove "JavaFX SDK" and "opencv" if they exist, and click "Add Library". Select "User Library" and import the OpenCV library. Click "Add Library" again to add the JavaFX SDK.

Now the error notifications should disappear.

3.2 Run the Skeleton Program

Click the "Run" button in eclipse's main window to run the program. The program should look like this:



The logic behind the skeleton is very simple. Once the user clicks the "Open" button, the programs loads "test.png" from the "resources" folder. Once the user clicks the "Play" button, the program plays (in audio) the image.

3.3 What You Should Do in This Assignment

Basic requirements:

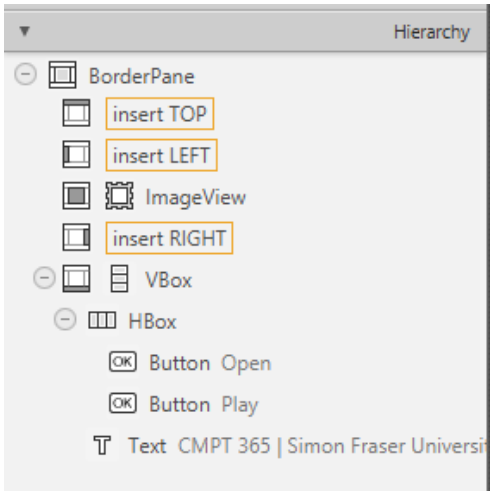
- Once the user clicks the “Open” button, the program should load a video rather than an image.
- The user should be able to select whatever file she wants.
- You should add a “click” sound between every two frames.

Optional:

- The GUI in the skeleton program has a fixed window size. Can you make it adjustable? Please note that the size of buttons, images, and texts should also be adjusted properly when the window size is modified.
- Currently, all the parameters (image width, height, sample rate, number of samples per column, etc.) are fixed. Can you modify the logic so that the user is allowed to specify different values? Can you do some experiments on what values give the best result?
- Can the volume be controllable?
- Can you output to a file that is readable by an audio player?
- ……

3.4 Edit the GUI

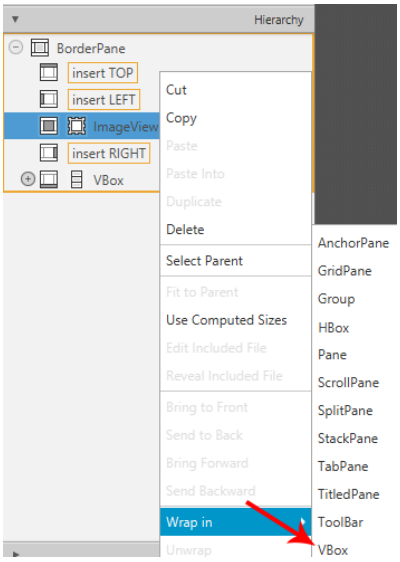
In the Project Explorer, right click “MainWindow.fxml” and select “Open with SceneBuilder”. At the bottom-left corner, click “Hierarchy” to expand it.



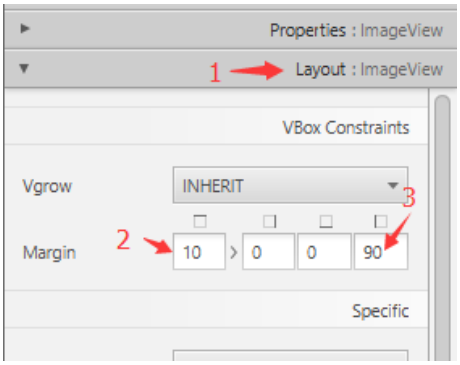
In case you haven't heard of the terms "pane" and "box" before, they literally mean "containers" – containers of other GUI components (buttons, texts, etc.). Containers are used to design the layout.

Read the hierarchy relation carefully and pay special attention to the icons and tags. You are smart enough to get a rough idea of how it works quickly.

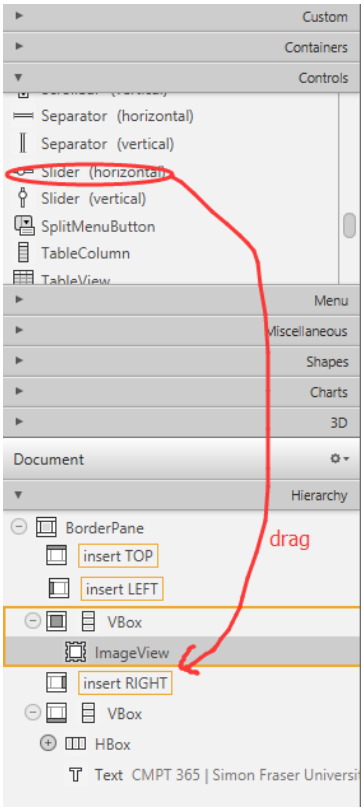
Suppose that we want to add a slider to our window right below the ImageView. In the "Hierarchy" panel, right click "ImageView" and select "Wrap in -> VBox".



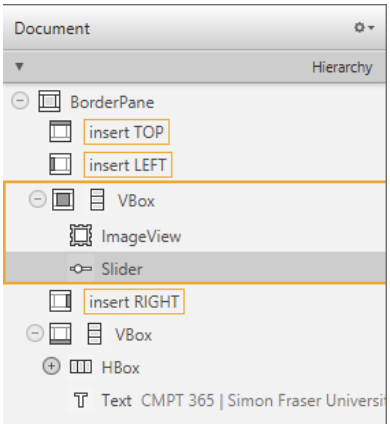
Select "ImageView" in the "Hierarchy" panel. Click "Layout" on the right to expand layout settings. Give the ImageView a margin of (top, 10), (left, 90) so that it is roughly in the middle of the window.



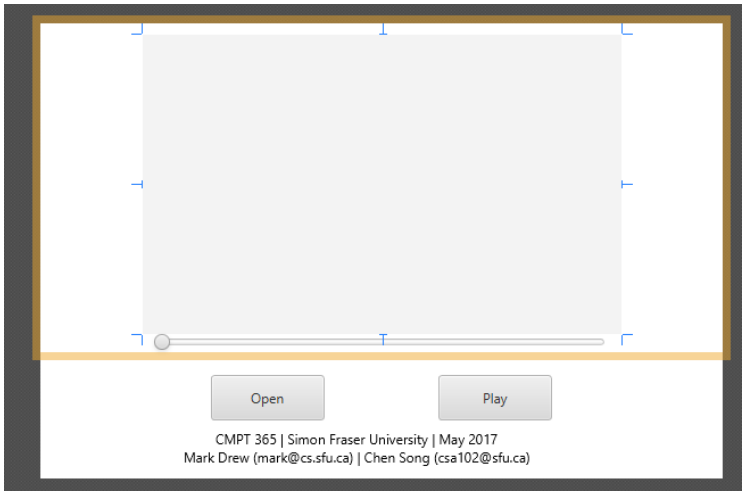
Select “Slider (horizontal)” in the “Controls” panel (located at the top-corner of the SceneBuilder). Drag it to the VBox we just created, and make sure it is after instead of before “ImageView”.



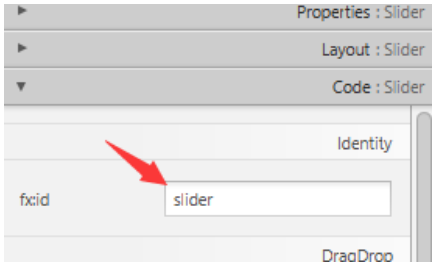
The “Hierarchy” panel should look like this now:



Select the slider we just created, and give it a margin of (right, 100), (left, 100) in the "Layout" panel. The final design should look like this:



Select the slider again. In the "Code" panel (below "Layout"), set the "fx:id" as "slider". This id allows our code to interact with the slider.

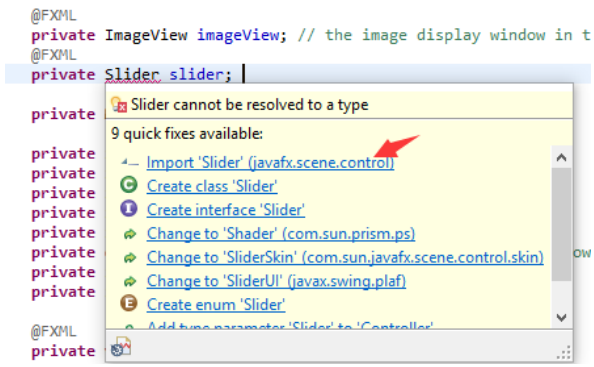


You should receive a warning saying that "No injectable field found in FXML Controller class for the id 'slider'". Ignore it for a moment.

Now switch back to eclipse's main window. Add the following code to the Controller class.

```
    @FXML
    private Slider slider;
```

You should receive an error saying that “Slider cannot be resolved to a type”. Click “Import ‘Slider’ (javafx.scene.control)” to fix it. Save the file.



Now the warning in SceneBuilder should disappear. In case it doesn’t, please restart SceneBuilder.

3.5 Display Video

In the getImageFilename() method, modify the logic so that it always returns the string “resources/test.mp4”. (You are smart enough to do it yourself.)

Add the following variables to the Controller class. A VideoCapture object is used to manipulate the video. You are smart enough to fix import errors.

```
        private VideoCapture capture;
        private ScheduledExecutorService timer;
```

Create a method using the following code.

```
protected void createFrameGrabber() throws InterruptedException {
  if (capture != null && capture.isOpened()) { // the video must be open
    double framePerSecond = capture.get(Videoio.CAP_PROP_FPS);

    // create a runnable to fetch new frames periodically
    Runnable frameGrabber = new Runnable() {
      @Override
      public void run() {
        Mat frame = new Mat();
          if (capture.read(frame)) { // decode successfully
            Image im = Utilities.mat2Image(frame);
            Utilities.onFXThread(imageView.imageProperty(), im);
            double currentFrameNumber =
capture.get(Videoio.CAP_PROP_POS_FRAMES);
            double totalFrameCount =
capture.get(Videoio.CAP_PROP_FRAME_COUNT);
            slider.setValue(currentFrameNumber / totalFrameCount *
(slider.getMax() - slider.getMin()));
          } else { // reach the end of the video
            capture.set(Videoio.CAP_PROP_POS_FRAMES, 0);
          }
```

```
      }
    };

    // terminate the timer if it is running
    if (timer != null && !timer.isShutdown()) {
      timer.shutdown();
      timer.awaitTermination(Math.round(1000/framePerSecond),
TimeUnit.MILLISECONDS);
    }

    // run the frame grabber
    timer = Executors.newSingleThreadScheduledExecutor();
    timer.scheduleAtFixedRate(frameGrabber, 0,
Math.round(1000/framePerSecond), TimeUnit.MILLISECONDS);
  }
}
```

Then modify the openImage() method to:

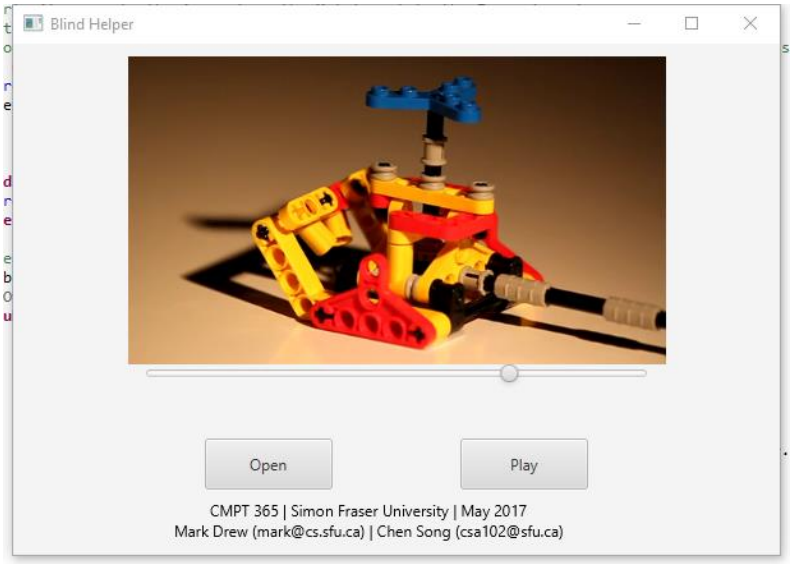
```
  capture = new VideoCapture(getImageFilename()); // open video file
  if (capture.isOpened()) { // open successfully
    createFrameGrabber();
  }
```

All these codes allow you to display a video repeated once you click the “Open” button. In case the createFrameGrabber() method looks funny to you, here I will briefly explain what is going on.

We want to be able to interact with the buttons while the video is playing. Therefore, the program must be able to handle user interactions and video display concurrently. To achieve this, we create another thread to handle video display, while the main thread is handling user interactions. The other thread is scheduled to run every several milliseconds (computed from the frame rate of your video).

To avoid potential conflicts between the two threads, we need to use the onFXThread() method to update the ImageView once we decode a new frame. You are encouraged to modify the code so that the image is passed to the ImageView without calling this method, and see what will happen.

Now the program should look like this. Can you modify the code and allow users to jump forwards and backwards by adjusting the slider?



4. Acknowledgements

The author of this tutorial is inspired by “OpenCV Java Tutorials” (http://opencv-java-tutorials.readthedocs.io/en/latest/03-first-javafx-application-with-opencv.html) written by Luigi De Russis. The skeleton program also contains “Utilities.java”, which is written by Luigi De Russis and Maximilian Zuleger.